Ritech Interface Board
RIF-171-1
（PCI バスタイプ）

取扱説明書

（第２版）



有限会社ライテックス



ジャストウェア株式会社

平成15年5月

目　次

1. 概要

本ボードは、PCI バス規格のショートサイズボードに、ハーフ実装の場合、アナログ入力 8 チャンネル、アナログ出力 8 チャンネル、デジタル入力 32 点、デジタル出力 32 点、PWM ジェネレータ 16 チャンネル、エンコーダ･カウンタ 8 チャンネルを、フル実装の場合、アナログ入力 16 チャンネル、アナログ出力 16 チャンネル、デジタル入力 32 点、デジタル出力 32 点、PWM ジェネレータ 16 チャンネル、エンコーダ･カウンタ 16 チャンネルを搭載したインターフェースボードである。

2. システム構成

2.1 入出力制御部

D/A コンバータとしては 13 ビット（ただし、使用ビットは 12 ビット）/ 8 チャンネルの AD7839(ANALOG DEVICES 社製)をハーフ実装時 1 個、フル実装時 2 個搭載している。A/D コンバータとしては 12 ビット/ 8 チャンネルの AD7891(ANALOG DEVICES 社製)をハーフ実装時 1 個、フル実装時 2 個搭載している。エンコーダ入力は 24 ビット/2 チャンネルの PCC160 （コスモテック社製）をハーフ実装時 4 個、フル実装時 8 個搭載している。デジタル入出力 32 点は TE7753（東京エレクトロン社製）を 1 個搭載している。PWM 出力 16 チャンネルは FPGA (ALTERA 社製) を 1 個搭載されている。

2.2 PCI バス･インターフェース部

本ボードでは、PCI バスのアドレス空間のうち、256 バイト分の領域を占有する。I/O アドレスはドライブを利用することでユーザーに知識がなくても使えるようにできている。

3. 仕様

表1～表7に仕様を示す。表8にボード全体の環境条件を示す。

表1. D/A 変換部の仕様

No	項　目	仕　様	備　考
1	チップ型式	AD7839(ANALOG DEVICES 社製)	
2	チップ使用数	2 個	フル実装時
3	出力チャンネル数	8×2 個＝16 チャンネル	フル実装時
4	分解能	13 ビット（ただし使用ビットは 12 ビット）	
5	変換時間	30μs(TYP)	
6	出力電圧範囲	±10V	
7	出力ドライブ電流	±10mA	
8	出力インピーダンス	0.5Ω(max)	

表2．A/D 変換部の仕様

No	項　　目	仕　　様	備　考
1	チップ型式	AD7891(ANALOG DEVICES 社製)	
2	チップ使用数	2 個	フル実装時
3	入力チャンネル数	8×2 個＝16 チャンネル	フル実装時
4	分解能	12 ビット	
5	変換時間	1.6μs(max)	
6	入力電圧範囲	±10V	
7	入力抵抗	15kΩ(min)	

表3．PWM ジェネレータ部の仕様

No	項　　目	仕　　様	備　考
1	チップ型式	EPF6016TC144-3(ALTERA 社製)	
2	チップ使用数	1個	
3	分解能	1/256	
4	チャンネル数	16 チャンネル	
5	周波数範囲	156Hz～40KHz	
6	デユーティー範囲	0%～100%	1%刻み
7	出力電流	TTL レベル　±25mA	

表4．エンコーダ部の仕様

No	項　　目	仕　　様	備　考
1	チップ型式	PCC160（コスモテックス社製）	
2	チップ使用数	8 個	フル実装時
3	分解能	24Bit（48Bit に拡張可能）	
4	チャンネル数	2×8 個＝16 チャンネル	フル実装時
5	周波数範囲	40MHz（最大）	
6	逓倍率	1,2,4	
7	入力モード	up/down パルス入力、A/B 相と Z 相入力可能	
8	入力	TTL、10kΩを通して 5V にプルアップ	

表6．デジタル入出力部の仕様

項	項　　目	仕　　様	備　考
1	チップ型式	TE7753（東京エレクトロン社製）	
2	チップ使用数	1	
3	入力チャンネル数	32 点	
4	出力チャンネル数	32 点	
5	入力インターフェース	TTL レベル	
6	出力インターフェース	TTL レベル　チャンネル 25～32（12mA）	
7	出力インターフェース	TTL レベル　その他（4mA）	
補足：　各ピンは 10kΩを通して 5V にプルアップされています。			

表5. PCI バス･インターフェース部の仕様

項	項　目	仕　様	備　考
1	アドレス デコード幅	下位 16 ビット	
2	アドレス空間 占有幅	256 バイト幅(ワード幅)	
3	データ サイズ	16 ビット長	
4	割り込み発生源	4 種類(詳細:ソフトウェアマニュアル参照)	

表7. 割り込みポート

項	項　目	仕　様	備　考
1	チャンネル数	4 チャンネル	
2	割り込みセンス方式	エッジ･レベルセンス選択可	

表8. ボード環境条件の仕様

項	項　目	仕　様	備　考
1	＋5V 消費電流	ハーフ実装時約 800mA、フル実装時約1A	
2	周囲温度	0～50℃(但し、結露しないこと)	

４．ボード外形とコネクタ割付

4.1 ボード外形

図1に示されているように、ボードの寸法(175mm×105mm)は PCI バス･ボード規格のショートサイズに準拠している。ボードの部品の高さは表で 14mm、裏で 8mm の厚さとなる。



図1.　ボードの外形写真

図2はボードの部品配置図を示している。

ボード表の部品配置

ボード裏の部品配置

図2.　ボードの部品配置図

4.2 コネクタ番号割付

表9にコネクタ番号と各チップのインターフェース信号の割付一覧を示している。なお、各コネクタの詳細なピン割付は付録に記載する。

表9. コネクタ番号割付一覧表

項	コネクタ	ピン数	接続チップ	信号種類	信号数
1	CN1	100	ENC1～8 PWM PIO	エンコーダ信号入力 PWM 信号出力 デジタル信号出力	16 チャンネル デジタル 32 点
2	CN2	100	AD1～2 DA1～2 PIO	アナログ信号入力 アナログ信号出力 デジタル信号入力	デジタル入力 32 点 16 チャンネル
3	CN3	7	INT	割り込み入力	4 チャンネル

5. モジュール構成

表10. モジュール構成表

項	品名	用途
1	RIF-171-1-A	フル実装タイプ
2	RIF-171-1-B	ハーフ実装タイプ

6. 梱包内容

1) インタフェースボード本体　1個
2) 取扱説明書　1 冊
3) ドライバソフト CD-ROM　1 枚
4) ケーブルとコネクタ(100ピン、50cm)　2 本
5) ユーザー登録カード　1 枚

注) もっと長いケーブルをご希望の方は別途お引き受け可能ですが、追加料金がかかります。

7. 保証範囲

1) Ritech Interface Board RIF-171-1 A/B の標準価格には、次の項目は含まれておりませんので、予め御了承おき下さい。
  (1) インタフェースボードとドライバソフトの適合性の検討。
  (2) 試運転と調整。
  (3) ボード故障時の現地調査と修理。
2) 保証要項

保証期間は納入後 6 ヶ月とします。この期間内で、使用上の注意が守られて、かつ、故障した場合には無償でこれを交換致しますが、ご返送いただく運送費は、ご負担下さい。

次のような場合には、保証期間内でも有償修理になります。

a. 使用上の誤り、或いは、不当改造や修理による故障及び損傷。

b. 落下、振動などによる損傷。

c. 火災、天災、塩害、ガス、異常電圧などによる故障、損傷の場合。

d. 接続している外部機器に起因して故障した場合。

e. 弊社以外の手で改造、修理がなされた場合。

3) 修理費用

有償修理については、次の規定に従って費用を請求させて戴きます。

修理費用＝ \40,000 ＋ 部品代 ＋ 運送費

故障の程度により上記の費用を超える場合もあります。又、程度により修理ができない場合もあります。

ご返送いただく運送費は、ご負担下さい。お預りする期間は、弊社到着後15営業日程度です。(部品在庫有する場合)

4) 点検費用

故障と思われてご返送いただき、弊社で調整した結果、異常が認められなかった場合は、保証期間の内外にかかわらず、下記の費用を請求させていただきます。

点検費用 ＝ \20,000 ＋ 運送費

ご返送いただく運送費はご負担下さい。お預りする期間は、弊社到着後15営業日程度です。

8. 付録

表11. CN1の端子割り付け

ピン	信号記号	信号名称	ピン	信号記号	信号名称
1	APhase1	エンコーダ CH1 の A 相入力	2	PWM1	PWM CH1 の出力
3	BPhase1	エンコーダ CH1 の B 相入力	4	PWM2	PWM CH2 の出力
5	ZPhase1	エンコーダ CH1 の Z 相入力	6	PWM3	PWM CH3 の出力
7	APhase2	エンコーダ CH2 の A 相入力	8	PWM4	PWM CH4 の出力
9	BPhase2	エンコーダ CH2 の B 相入力	10	PWM5	PWM CH5 の出力
11	ZPhase2	エンコーダ CH2 の Z 相入力	12	PWM6	PWM CH6 の出力
13	APhase3	エンコーダ CH3 の A 相入力	14	PWM7	PWM CH7 の出力
15	BPhase3	エンコーダ CH3 の B 相入力	16	PWM8	PWM CH8 の出力
17	ZPhase3	エンコーダ CH3 の Z 相入力	18	PWM9	PWM CH9 の出力
19	APhase4	エンコーダ CH4 の A 相入力	20	PWM10	PWM CH10 の出力
21	BPhase4	エンコーダ CH4 の B 相入力	22	PWM11	PWM CH11 の出力
23	ZPhase4	エンコーダ CH4 の Z 相入力	24	PWM12	PWM CH12 の出力
25	GND	グランド	26	PWM13	PWM CH13 の出力
27	APhase5	エンコーダ CH5 の A 相入力	28	PWM14	PWM CH14 の出力
29	BPhase5	エンコーダ CH5 の B 相入力	30	PWM15	PWM CH15 の出力
31	ZPhase5	エンコーダ CH5 の Z 相入力	32	PWM16	PWM CH16 の出力
33	APhase6	エンコーダ CH6 の A 相入力	34	GND	グランド
35	BPhase6	エンコーダ CH6 の B 相入力	36	OUT1	デジタル出力 1
37	ZPhase6	エンコーダ CH6 の Z 相入力	38	OUT2	デジタル出力 2
39	APhase7	エンコーダ CH7 の A 相入力	40	OUT3	デジタル出力 3
41	BPhase7	エンコーダ CH7 の B 相入力	42	OUT4	デジタル出力 4
43	ZPhase7	エンコーダ CH7 の Z 相入力	44	OUT5	デジタル出力 5
45	APhase8	エンコーダ CH8 の A 相入力	46	OUT6	デジタル出力 6
47	BPhase8	エンコーダ CH8 の B 相入力	48	OUT7	デジタル出力 7
49	ZPhase8	エンコーダ CH8 の Z 相入力	50	OUT8	デジタル出力 8
51	APhase9	エンコーダ CH9 の A 相入力	52	OUT9	デジタル出力 9
53	BPhase9	エンコーダ CH9 の B 相入力	54	OUT10	デジタル出力 10
55	ZPhase9	エンコーダ CH9 の Z 相入力	56	OUT11	デジタル出力 11
57	APhase10	エンコーダ CH10 の A 相入力	58	OUT12	デジタル出力 12
59	BPhase10	エンコーダ CH10 の B 相入力	60	OUT13	デジタル出力 13
61	ZPhase10	エンコーダ CH10 の Z 相入力	62	OUT14	デジタル出力 14
63	APhase11	エンコーダ CH11 の A 相入力	64	OUT15	デジタル出力 15
65	BPhase11	エンコーダ CH11 の B 相入力	66	OUT16	デジタル出力 16
67	ZPhase11	エンコーダ CH11 の Z 相入力	68	OUT17	デジタル出力 17
69	APhase12	エンコーダ CH12 の A 相入力	70	OUT18	デジタル出力 18
71	BPhase12	エンコーダ CH12 の B 相入力	72	OUT19	デジタル出力 19
73	ZPhase12	エンコーダ CH12 の Z 相入力	74	OUT20	デジタル出力 20
75	GND	グランド	76	OUT21	デジタル出力 21
77	APhase13	エンコーダ CH13 の A 相入力	78	OUT22	デジタル出力 22
79	BPhase13	エンコーダ CH13 の B 相入力	80	OUT23	デジタル出力 23
81	ZPhase13	エンコーダ CH13 の Z 相入力	82	OUT24	デジタル出力 24
83	APhase14	エンコーダ CH14 の A 相入力	84	OUT25	デジタル出力 25
85	BPhase14	エンコーダ CH14 の B 相入力	86	OUT26	デジタル出力 26
87	ZPhase14	エンコーダ CH14 の Z 相入力	88	OUT27	デジタル出力 27
89	APhase15	エンコーダ CH15 の A 相入力	90	OUT28	デジタル出力 28
91	BPhase15	エンコーダ CH15 の B 相入力	92	OUT29	デジタル出力 29
93	ZPhase15	エンコーダ CH15 の Z 相入力	94	OUT30	デジタル出力 30
95	APhase16	エンコーダ CH16 の A 相入力	96	OUT31	デジタル出力 31
97	BPhase16	エンコーダ CH16 の B 相入力	98	OUT32	デジタル出力 32
99	ZPhase16	エンコーダ CH16 の Z 相入力	100	GND	グランド

表１２．CN2 の端子割り付け

ピン	信号記号	信号名称	ピン	信号記号	信号名称
1	AD1	アナログ入力 CH1	2	DA10	アナログ出力 CH10
3	ADG1	アナロググランド CH1	4	DAG10	アナロググランド CH10
5	AD2	アナログ入力 CH2	6	DA11	アナログ出力 CH11
7	ADG2	アナロググランド CH2	8	DAG11	アナロググランド CH11
9	AD3	アナログ入力 CH3	10	DA12	アナログ出力 CH12
11	ADG3	アナロググランド CH3	12	DAG12	アナロググランド CH12
13	AD4	アナログ入力 CH4	14	DAG13	アナログ出力 CH13
15	ADG4	アナロググランド CH4	16	DAG13	アナロググランド CH13
17	AD5	アナログ入力 CH5	18	DAG14	アナログ出力 CH14
19	ADG5	アナロググランド CH5	20	DAG14	アナロググランド CH14
21	AD6	アナログ入力 CH6	22	DAG15	アナログ出力 CH15
23	ADG6	アナロググランド CH6	24	DAG15	アナロググランド CH15
25	AD7	アナログ入力 CH7	26	DAG16	アナログ出力 CH16
27	ADG7	アナロググランド CH7	28	DAG16	アナロググランド CH16
29	AD8	アナログ入力 CH8	30	GND	グランド
31	ADG8	アナロググランド CH8	32	GND	グランド
33	AD9	アナログ入力 CH9	34	IN1	デジタル入力 1
35	ADG9	アナロググランド CH9	36	IN2	デジタル入力 2
37	AD10	アナログ入力 CH10	38	IN3	デジタル入力 3
39	ADG10	アナロググランド CH10	40	IN4	デジタル入力 4
41	AD11	アナログ入力 CH11	42	IN5	デジタル入力 5
43	ADG11	アナロググランド CH11	44	IN6	デジタル入力 6
45	AD12	アナログ入力 CH12	46	IN7	デジタル入力 7
47	ADG12	アナロググランド CH12	48	IN8	デジタル入力 8
49	AD13	アナログ入力 CH13	50	IN9	デジタル入力 9
51	ADG13	アナロググランド CH13	52	IN10	デジタル入力 10
53	AD14	アナログ入力 CH14	54	IN11	デジタル入力 11
55	ADG14	アナロググランド CH14	56	IN12	デジタル入力 12
57	AD15	アナログ入力 CH15	58	IN13	デジタル入力 13
59	ADG15	アナロググランド CH15	60	IN14	デジタル入力 14
61	AD16	アナログ入力 CH16	62	IN15	デジタル入力 15
63	ADG16	アナロググランド CH16	64	IN16	デジタル入力 16
65	DA1	アナログ出力 CH1	66	GND	グランド
67	DAG1	アナロググランド CH1	68	IN17	デジタル入力 17
69	DA2	アナログ出力 CH2	70	IN18	デジタル入力 18
71	DAG2	アナロググランド CH2	72	IN19	デジタル入力 19
73	DA3	アナログ出力 CH3	74	IN20	デジタル入力 20
75	DAG3	アナロググランド CH3	76	IN21	デジタル入力 21
77	DA4	アナログ出力 CH4	78	IN22	デジタル入力 22
79	DAG4	アナロググランド CH4	80	IN23	デジタル入力 23
81	DA5	アナログ出力 CH5	82	IN24	デジタル入力 24
83	DAG5	アナロググランド CH5	84	IN25	デジタル入力 25
85	DA6	アナログ出力 CH6	86	IN26	デジタル入力 26
87	DAG6	アナロググランド CH6	88	IN27	デジタル入力 27
89	DA7	アナログ出力 CH7	90	IN28	デジタル入力 28
91	DAG7	アナロググランド CH7	92	IN29	デジタル入力 29
93	DA8	アナログ出力 CH8	94	IN30	デジタル入力 30
95	DAG8	アナロググランド CH8	96	IN31	デジタル入力 31
97	DA9	アナログ出力 CH9	98	IN32	デジタル入力 32
99	DAG9	アナロググランド CH9	100	GND	グランド

表13. CN3 の端子割り付け

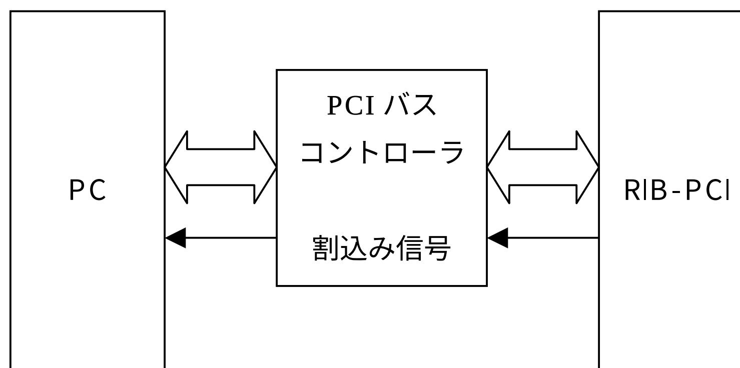
ピン	信号記号	信号名称
1	INT1	外部割込み入力 CH1
2	INT2	外部割込み入力 CH2
3	INT3	外部割込み入力 CH3
4	INT4	外部割込み入力 CH4
5	WDT	ウォッチドッグタイマー出力
6	GND	グランド
7	NC	何も接続しない

第二部 ソフトウェアマニュアル

１．はじめに

PCI バスタイプ Ritech Interface Board（以下 RIB-PCI と略称）には、１６チャンネル AD コンバータ、１６チャンネル DA コンバータ、１６チャンネル PWM、１６チャンネルカウンター、３２チャンネル PI、３２チャンネル PO、４チャンネルの外部割込みが集約されている。RIB-PCI にはマイコンが内蔵されていないので、PC 側のアプリケーションが PCI バスコントローラを通じてインタフェース IC チップを直接に制御しなければならない（図 1-1)。即ち、アプリケーションがそれぞれの IC チップの内蔵レジスタに直接指示データを書き込んだり、結果データを読みこんだりする必要がある。IC チップによって内蔵レジスタのロケーションや内容定義やアクセス方法などが全く異なるので、RIB-PCI 上の全ての IC チップのマニュアルとデータシートを読んで理解しなければならない。但し、RIB-PCI ボードには高性能のドライバソフトが添付されている。そのドライバソフトを利用すれば、IC チップのマニュアルやデータシートを読む必要がなく、インタフェース IC の制御が非常に簡単になる。詳しくは第 3 章をご覧ください。

図 1-1



PCI ボードの IO アドレスは、PC が立ち上がるときに BIOS によって自動的に決められる。ドライバソフトを利用すれば、RIB-PCI の IO アドレスを知る必要がないが、直接アクセスする場合は RIB-PCI の IO アドレスを知らなければならない。ドライバソフトが RIB-PCI の IO アドレスを取得する API 関数も提供している。RIB-PCI の IO アドレスが分かれば、各インタフェース IC のアドレスが次の通りになる。

表１-１

インタフェース IC	IC アドレス
AD コンバータ（CH０１～０８）	ベースアドレス＋０ｘ００
AD コンバータ（CH０９～１６）	ベースアドレス＋０ｘ１０
DA コンバータ（CH０１～０８）	ベースアドレス＋０ｘ２０
DA コンバータ（CH０９～１６）	ベースアドレス＋０ｘ３０
カウンター（CH０１～０２）	ベースアドレス＋０ｘ４０
カウンター（CH０３～０４）	ベースアドレス＋０ｘ４８
カウンター（CH０５～０６）	ベースアドレス＋０ｘ５０
カウンター（CH０７～０８）	ベースアドレス＋０ｘ５８

PIO（CH０１～３２）	ベースアドレス＋０ｘ６０
PWM（CH０１～１６）	ベースアドレス＋０ｘ８０
カウンター（CH０９～１０）	ベースアドレス＋０ｘE０
カウンター（CH１１～１２）	ベースアドレス＋０ｘE８
カウンター（CH１３～１４）	ベースアドレス＋０ｘF０
カウンター（CH１５～１６）	ベースアドレス＋０ｘF８

ここでベースアドレスは RIB-PCI ボードの IO アドレスで、IC アドレスは、IC 内蔵レジスタのスタートアドレスである。各内蔵レジスタのアドレスとスタートアドレスとの関係は、IC のマニュアルをご参考ください。これらのマニュアルが製品には添付されていないので、必要な場合、弊社の技術サポートの方にご連絡下さい。

RIB-PCI は、４チャンネルの外部割込みを発生する以外、他の割り込みを一切発生しない。４チャンネルの割り込みが一つの割り込み番号を共用する。PCI ボードの共用割り込み番号も PC が立ち上がるときに BIOS によって自動的に決められる。ドライバソフトを利用する場合、割り込み番号を知らなくてもいいが、ドライバソフトが割り込み番号を取得する API 関数も提供している。

RIB-PCI の IO アドレスと割り込み番号は BIOS によって自動的に割り付けられるので、他の IO ボードのアドレスと割込み番号と衝突する可能性がほとんどない。また、RIB-PCI を二枚以上同時に使う場合も、IO アドレスと割込み番号が重複するような心配も要らない。

RIB-PCI を PC に取り付けて利用する前に、まず付属の CD-ROM のフォルダ「\Win９８\Test」にあるテストプログラムを実行して、ボード上の全てのインタフェースチップが正常に稼動しているかどうかを確認することをお勧めします。

２．Windows ドライバ・ソフト仕様

この節は、Windows 95/98 を利用しているユーザーが RIB-PCI を簡単にアクセスできるために提供されているドライバソフトについて説明する。Windows 95/98 では、アプリケーションが直接ハードウェアのリソースをアクセスするのではなく、ドライバソフトを通じてアクセスする方法が推薦されている。特に割込みイベントをキャッチするのはドライバソフトしかできない。従って、本ボードもドライバソフトを提供している。

一、Windows ドライバのファイル構成

ドライバソフトは次のファイルから構成されている。

ファイル名	役割説明
Jwribpci.inf	ドライバのインストール情報が格納されている。
Jwribpci.vxd	RIB-PCI をアクセスするドライバ本体である。
Jwribpci.dll	アプリケーションとのインタフェースを提供する動的ライブラリで、API 関数が含まれる。
Jwribpci.lib	API 関数のリンク情報を含める静的なライブラリで、アプリケーションとリンクする必要がある。
Jwribpci.h	API 関数のプロトタイプが含まれている。アプリケーションのソースプログラムにインクルードする必要がある。

上記のファイルが付属の CD-ROM のフォルダ「\Win98\Driver」に格納されている。これらのプログラムが全て３２ビットなので、３２ビットのアプリケーションがしか利用できない。

二、Windows ドライバのインストール

RIB-PCI ボードを PC に取り付けて、PC を起動すると、本製品がプラグ＆プレイで認識され、新しいハードウェアの追加ウィザードが起動する。以下の手順に従ってドライバをインストールして下さい。

●「次の新しいドライバを検索しています」が表示されている画面で＜次へ＞ボタンを押す。
●「検索方法を選択して下さい」が表示されている画面で、「使用中のデバイスに最適なドライバを検索する（推奨)」を選んで＜次へ＞ボタンを押す。
●「検索場所の指定」に「D:\Win98\Driver」を入力して＜次へ＞ボタンを押す。ここで、「D」は CD-ROM のドライブとする。
●PC が起動したら、付属の CD-ROM のフォルダ「\Win98\Driver」に格納されているファイル「Jwribpci.vxd」と「Jwribpci.dll」をアプリケーションと同じフォルダにコピーするか、Windows のシステムフォルダ（通常「C:\Windows\System」）にコピーする。
●最後に付属の CD-ROM のフォルダ「\Win98\Driver」に格納されているファイル「Jwribpci.lib」と「Jwribpci.h」をアプリケーションのソースフォルダにコピーする。

以上、ドライバのインストールが終わる。

三、Windows ドライバの利用方法

　Windows ドライバは、アプリケーションが簡単にインタフェースボードをアクセスするために、１１種類の API関数を用意している。アプリケーションが Jwribpci.lib とリンクすれば、API 関数を使って RIB-PCI を制御することができる。以降、これらの API 関数について詳しく説明する。

１．アドレス取得関数

関数名	**int GetRegAdrEx(int iRibNo)**
機　能	RIB-PCI ボードの IO アドレスを取得する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる）
戻り値	**Not -1:** IO アドレス　　**-1:** 異常終了

２．割込み番号取得関数

関数名	**int GetIrqNoEx(int iRibNo)**
機　能	RIB-PCI が使う割込みの番号を取得する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる）
戻り値	**Not -1:** 割込み番号　　**-1:** 異常終了

３．割込みイベントメッセージ設定関数

関数名	**int SetWndMsgEx(int iRibNo, HANDLE hWnd, UINT uMsg)**
機　能	RIB-PCIに割込みが発生したときにWindowsアプリケーションに知らせるためのメッセージを設定する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **hWnd:** メッセージを受け取る Window のハンドル **uMsg:** 割込みイベントに対応するメッセージ（0xC000 以上の整数）
戻り値	**0:** 正常終了　　**Not 0:** 異常終了

４．割込みイベントメッセージ許可関数

関数名	**int EnableIntMsgEx(int iRibNo, BOOL bEnable)**
機　能	割込みイベントが発生したら Windows メッセージを生成するかどうかを設定する。 割込みを禁止するかどうかを設定する関数でないことにご注意下さい。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **bEnable:** メッセージ許可フラグ（TRUE：許可、FALSE：禁止）
戻り値	**0:** 正常終了　　**Not 0:** 異常終了

5．パラメータのないコマンド送信関数

関数名	**BOOL SendCmdEx(int iRibNo, int iChNo, WORD wCmd)**
機　能	パラメータを持っていないコマンドを RIB-PCI に送信する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **iChNo:** チャンネル番号（0 から始まる） **wCmd:** 送信するコマンド（第３章をご参考下さい）
戻り値	**TRUE:** 正常終了　　**FALSE:** 異常終了

6．パラメータのあるコマンド送信関数

関数名	**BOOL SendCmdDataEx(int iRibNo, int iChNo, WORD wCmd, WORD wData)**
機　能	パラメータを持っているコマンドを RIB-PCI に送信する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **iChNo:** チャンネル番号（0 から始まる） **wCmd:** 送信するコマンド（第３章をご参考下さい） **wData:** コマンドが持っているパラメータ
戻り値	**TRUE:** 正常終了　　**FALSE:** 異常終了

7．ダブルワード・パラメータのあるコマンド送信関数

関数名	**BOOL SendCmdDWDataEx(int iRibNo, int iChNo, WORD wCmd, DWORD dwData)**
機　能	ダブルワード・パラメータを持っているコマンドを RIB-PCI に送信する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **iChNo:** チャンネル番号（0 から始まる） **wCmd:** 送信するコマンド（第３章をご参考下さい） **dwData:** コマンドが持つダブルワード・パラメータ（例えばカウンターの値）
戻り値	**TRUE:** 正常終了　　**FALSE:** 異常終了

8．データ受信関数

関数名	**BOOL ReadDataEx(int iRibNo, WORD *pwData, int iCount)**
機　能	RIB-PCI からデータを受信する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **pwData:** 受信データを受取るバッファーのポイント。バッファーの長さは **iCount** ワード以上でなければならない。 **iCount:** 受信データのワード数
戻り値	**TRUE:** 正常終了　　**FALSE:** 異常終了

9．ダブルワード・データ受信関数

関数名	**BOOL ReadDWDataEx(int iRibNo, DWORD *pdwData)**
機　能	RIB-PCI からダブルワード・データを受信する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **pdwData:** ダブルワード・受信データ（例えばカウンターの値）を受取るバッファーのポイント。バッファーの長さはダブルワードである。
戻り値	**TRUE:** 正常終了　　**FALSE:** 異常終了

10．割り込み時のコールバック関数の設定関数

関数名	**int SetVxdCallBackEx(int iRibNo, RIBVXDCALLBACK *pUserProc, DWORD dwCallBackSize, void *pUserData, DWORD dwUserDataSize)**
機　能	割り込み発生時に自動的に呼び出される関数（割り込み処理関数）を設定する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **pUserProc:** 割り込み処理関数（コールバック関数）のポインタ **dwCallBackSize:** コールバック関数のサイズ。正確にわからない場合に多めに設定する。 **pUserData:** 割り込み処理に必要なデータを格納するバッファーのポインタ **dwUserDataSize:** データバッファーのサイズ
戻り値	**0:** 正常終了　　**Not 0:** 異常終了

11．ボード番号取得関数

関数名	**int GetRibNoEx(int iRibID)**
機　能	RIB-PCI ボードの番号を取得する。
引　数	**iRibID:** RIB-PCI ボードの ID（製造シリアル番号）
戻り値	**Not -1:** ボード番号　　**-1:** 異常終了

補足説明：

●引数 iRibNo は RIB-PCI ボードの番号を各関数に渡す。1 枚の RIB-PCI のみを使う場合、番号を常に０に設定すればいい。２枚以上の RIB-PCI を同時に使う場合、GetRibNoEx 関数を使って、RIB-PCI の製造シリアル番号の下 4 桁を引数として渡して、当該ボードの番号を取得してください。PC の拡張ボードの構成によって番号が変わる可能性があるので、各 RIB-PCI ボードの番号を定数として定義するのではなく、アプリケーションの初期化ルーチンに GetRibNoEx を呼び出して動的にボードの番号を決めてください。

●ISA バスタイプの RIB ボードとの互換性を維持するために、ドライバソフトが RIB-ISA のマイコンの機能をシミュレートする。即ち、ドライバソフトがマイコンの代わりに、アプリケーションからのコマンドを受け付けて解釈して実行する。従って、RIB-PCI を制御するに

は第３章に説明されている全ての RIB-ISA コマンドがそのまま使える。ドライバソフトも、SendCmdEx, SendCmdDataEx, ReadDataEx という３つの API 関数を用意して、これらの関数を利用すれば、ドライブへコマンドを送ってたり、ドライバからデータを受け取ったりすることができる。SendCmdDWDataEx, ReadDWDataEx が２４ビットカウンタをアクセスするために追加された関数です。

●引数 iChNo はチャンネル番号である。RIB-PCI では各種インタフェースがそれぞれ複数のチャンネルを持っているので、インタフェースをアクセスする多くのコマンドはチャンネルを指定しなければならない。チャンネルを指定する必要のないコマンドの場合、チャンネル番号を０にしてください。また、チャンネル番号が０番から始まるので、チャンネル N をアクセスしたい場合、iChNo に N-1 をセットしなければならない。

●使いたい機能が標準コマンドでサポートしていない場合、又はコマンドを利用せずに直接 IC のレジスタをアクセスしたい場合、GetRegAdr と GetIrqNo を使って RIB-PCI の IO アドレスと割り込み番号を取得してください。

●外部割込みが発生するとき、ドライバソフトがそれを検出し、外部割込み番号（１から始まる）をパラメータ wParam のローバイトにセットしてメッセージ uMsg と一緒にアプリケーションに送る。従って、アプリケーション側が uMsg を受け取ったら wParam によって外部割込み番号を知り、対応する処理プログラムを実行すればいいです。二枚以上の RIB-PCI を利用する場合、それぞれ異なる割込みイベントメッセージを設定すれば、どの RIB-PCI に割込みが発生したかを判断できる。一方、パラメータ lParam に割込みの発生回数をセットしている。

●Windows メッセージによる割込み処理がどうしても遅くて間に合わない場合、CallBack 関数を利用して、割り込み発生時に必要最低限の処理を CallBack 関数に移せば、割込みに対する高速の反応が可能になる。CallBack 関数の定義が次の通りである。

```
typedef int (RIBVXDCALLBACK)(WORD wIntNo, BOOL* bEnableMessage,
                void* pUserData, DWORD dUserDataSize);
```

ここで、wIntNo は外部割込み番号で、ドライバが CallBack 関数に渡す。bEnableMessage は、割込み発生に対応するメッセージをアプリケーションに送信するかどうかを決めるフラグで、CallBack 関数がドライバに渡す。TRUE の場合、ドライバが割込みメッセージをアプリに送信し、FALSE の場合、送信しないことになる。 PUserData と dUserDataSize は 関数 SetVxdCallBackEx により渡されたデータバッファーのポインタとサイズをドライバがそのまま CallBack 関数に渡す。CallBack 関数で処理したいデータを SetVxdCallBackEx により渡すことになる。

●アプリケーションが一度一つの外部割込みをしか処理できないので、RIB-PCI では割込みが発生すると、割込み発生フラグ INT がセットされ、次の割込みを待たせる。従って、アプリケーション側が割込み処理を完了したら、速やかにコマンド INT_Clear をドライバに送信

してフラグ INT をクリアしなければならない。詳しくは第 3 章のコマンド説明をご参照下さい。

付属 CD-ROM のフォルダ「\Win98\Sample」にドライバが提供している API 関数の使い方を示すサンプルプログラムが格納されているので、ご参考下さい。サンプルプログラムをコンパイルしたい場合、マイクロソフトの Visual C++ Ver.5.0 以上を使ってください。

３．コマンド仕様

RIB-PCI は、ユーザーのアプリケーションが簡単に入出力インタフェースをアクセスできるために、表３-１に示されている２２種類のコマンドを提供する。

表３-１

ｺｰﾄﾞ	記　号	説　　明	ﾁｬﾝﾈﾙ	渡すﾃﾞｰﾀ	貰うﾃﾞｰﾀ
10H	AD_Read	ＡＤからﾃﾞｰﾀを読み込む	あり	なし	変換結果
20H	DA_Write	ＤＡへﾃﾞｰﾀを書き込む	あり	変換ﾃﾞｰﾀ	なし
30H	PWM_Start	ＰＷＭ出力を開始する	あり	なし	なし
31H	PWM_Stop	ＰＷＭ出力を停止する	あり	なし	なし
32H	PWM_Rate	出力ﾊﾟﾙｽの周波数を設定する	あり	周波数	なし
33H	PWM_Duty	出力ﾊﾟﾙｽのﾃﾞｭｰﾃｨｰを設定する	あり	ﾃﾞｭｰﾃｨｰ	なし
40H	ENC_Start	ｴﾝｺｰﾀﾞｰｶｳﾝﾀを開始する	あり	なし	なし
41H	ENC_Stop	ｴﾝｺｰﾀﾞｰｶｳﾝﾀを停止する	あり	なし	なし
42H	ENC_Mode	入力ﾓｰﾄﾞと逓倍数を設定する	あり	ﾓｰﾄﾞ/逓倍数	なし
43H	ENC_Read	ｶｳﾝﾀの値を読み出す	あり	なし	ｶｳﾝﾀ値
44H	ENC_Clear	ｶｳﾝﾀの値をｸﾘｱする	あり	なし	なし
45H	ENC_Zphase	ｴﾝｺｰﾀﾞｰＺ相の動作を設定する	なし	ﾁｬﾝﾈﾙ・ﾌﾗｸﾞ	なし
46H	ENC_Zploe	ｴﾝｺｰﾀﾞｰＺ相の極性を設定する	なし	極性	なし
47H	ENC_Set	ｶｳﾝﾀの値を設定する	あり	ｶｳﾝﾀ値	なし
50H	PIO_Read	１６Bit 単位でﾎﾟｰﾄから入力する	あり	なし	入力ﾃﾞｰﾀ
51H	PIO_Write	１６Bit 単位でﾎﾟｰﾄへ出力する	あり	出力ﾃﾞｰﾀ	なし
52H	PIO_Set	指定ﾎﾟｰﾄをﾊｲにｾｯﾄする	あり	なし	なし
53H	PIO_Clear	指定ﾎﾟｰﾄをﾛｰにｾｯﾄする	あり	なし	なし
60H	INT_Enable	外部割込みを許可する	あり	なし	なし
61H	INT_Disable	外部割込みを禁止する	あり	なし	なし
62H	INT_Mode	外部割込みﾄﾘｶﾞｰﾓｰﾄﾞを設定する	あり	ﾄﾘｶﾞｰﾓｰﾄﾞ	なし
63H	INT_Clear	割込みﾌﾗｸﾞをｸﾘｱする	なし	なし	なし
FFH	ALL_Reset	IO ﾎﾞｰﾄﾞをﾘｾｯﾄし初期状態に戻す	なし	なし	なし

以下、これらのコマンドについて詳しく説明する。

(1) AD_Read

これは AD コンバーターから変換データを読み込むコマンドである。ドライバがこのコマンドを受信したら、直ちに AD 変換を開始し、変換終了を待つ。AD 変換が終了したらアプリケーションに戻る。戻ってきたらアプリケーションが直ちに ReadDataEx 関数を呼び出して変換結果を取得してください。AD 変換が 1.6μs しかかからないので、このコマンドが非常に効率が良い、使いやすいコマンドである。

変換結果は 12Bit のデータで、0000H から 0FFFH までの範囲に入る。変換データ D と電圧 V との関係は次の式で計算できる。

D＝INT(V×4096.0／20.0＋2047.5)

ここで、INT は整数部のみを取る関数を示す。

例　12 チャンネルの AD 変換データを読み込みたい場合、次のようにを書けばいい。

```
SendCmdEx(0, 11, 0x10);   ReadDataEx(0, &wData,1);
```

(2) DA_Write

これは DA コンバーターへ変換データを書き込むコマンドである。ドライバがこのコマンドを受信したら、直ちに変換データを DA にセットし変換を開始する。

変換データは 12Bit で、0000H から 0FFFH までの範囲に入る。電圧 V と変換データ D との関係は次の式で計算できる。

V＝(D－2048.0)×20.0／4096.0

初期状態では全ての DA の出力が０Ｖになっている。

例　12 チャンネルの DA に 1V を出力したい場合、次のように書けばいい。

```
SendCmdDataEx(0, 11, 0x20, 0x08CD);
```

(3) PWM_Start

このコマンドは指定したチャンネルの PWM に対してパルス信号出力の開始を指示する。パルス・デューティーが固定の 50%であり、設定できない。

例　チャンネル 4 の PWM を開始させたい場合、次のように書けばいい。

```
SendCmdEx(0, 3, 0x30);
```

(4) PWM_Stop

このコマンドは指定したチャンネルの PWM に対してパルス信号出力の停止を指示し、出力をハイ・インピーダンスにする。初期状態では全ての PWM の出力がハイ・インピーダンスになっている。

例　チャンネル 4 の PWM を停止させたい場合、次のように書けばいい。

```
SendCmdEx(0, 3, 0x31);
```

(5) PWM_Rate

このコマンドは指定したチャンネルの PWM に対してパルス信号の周波数を設定する。16 チャンネルの PWM の周波数をそれぞれ設定できる。設定可能な範囲は 156.25Hz～40kHz である。ドライバに渡すデータ D（0～255）とパルス周波数 F の関係は次のようになっている。

F＝40000／(256-D)（Hz）

デフォルト周波数は、F＝156.25Hz（D＝0）になっている。

例　チャンネル 4 の周波数を 2kHz に設定したい場合、次のように書けばいい。

```
SendCmdDataEx(0, 3, 0x32, 0xEC);
```

(6) PWM_Duty

このコマンドは指定したチャンネルの PWM に対してパルス信号のデューティーを設定する。16 チャンネルの PWM のデューティーをそれぞれ設定できる。ドライバに渡すデータはパルス・デューティーである。デューティーの設定可能な範囲は 0～100 である。100 以上の数値を渡しても 100%のデューティーになる。

例　チャンネル 4 の PWM のデューティーを 16%に設定したい場合、次のように書けばいい。

```
SendCmdDataEx(0, 3, 0x33, 0x10);
```

(7) ENC_Start

このコマンドは指定したチャンネルのカウンタにカウントの開始を指示する。

例　チャンネル 3 のカウンタを開始させたい場合、次のように書けばいい。

```
SendCmdEx(0, 2, 0x40);
```

(8) ENC_Stop

このコマンドは指定したチャンネルのカウンタにカウントの停止を指示する。

例　チャンネル 3 のカウンタを停止させたい場合、次のように書けばいい。

```
SendCmdEx(0, 2, 0x41);
```

(9) ENC_Mode

このコマンドは指定したチャンネルのカウンタに対し、入力モードと逓倍数を設定する。ドライバに渡すデータと入力モード・逓倍数との関係は表３－２のように示す。デフォルトの入力モードは「1 相入力、Up／Down カウント」になっている。

例　チャンネル 3 のカウンタを 2 相入力、4 逓倍、差異カウントモードに設定したい場合、次のように書けばいい。

```
SendCmdEx(0, 2, 0x42, 0x03);
```

表 3-2

渡すデータ	入力モード・逓倍数
0	1 相入力、Up／Down カウント
1	2 相入力、1 逓倍、差異カウント
2	2 相入力、2 逓倍、差異カウント
3	2 相入力、4 逓倍、差異カウント

(10) ENC_Read

このコマンドは指定したチャンネルのカウンタからカウント値を読み出す。コマンドを送信した後に、ReadDWDataEx により２４ビットのカウンタ値を読み出す。

例　チャンネル 3 のカウンタ値を読み出したい場合、次のように書けばいい。

SendCmdEx(0, 2, 0x43);　ReadDWDataEx(0, &dwData);

(11) ENC_Clear

このコマンドは指定したチャンネルのカウンタをクリアする（カウント値＝０）。

例　チャンネル 3 のカウンタをクリアしたい場合、次のように書けばいい。

SendCmdEx(0, 2, 0x44);

(12) ENC_Zphase

このコマンドは 16 チャンネルのエンコーダーの Z 相信号に対して、Z 相信号が入力される（ハイからローに又はローからハイに変わる）とカウンタの値を自動クリアするかどうかを設定する。Z 相信号の極性はコマンド ENC_Zpole で設定する。マイコンに渡すデータは 16Bit で､1Bit は１チャンネルのエンコーダーと対応する。Bit0 はチャンネル 1 と、Bit15 はチャンネル１６と対応する。ビットをセットすると、自動クリアを許可し、ビットをクリアすると、自動クリアを禁止する。デフォルトでは全ての Z 相の自動クリアが禁止されている状態になっている。

例　チャンネル１、2 の Z 相自動クリアを許可し、それ以外の Z 相自動クリアを禁止したい場合、次のように書けばいい。

SendCmdDataEx(0, 0, 0x45, 0x0003);

(13) ENC_Zpole

このコマンドは 16 チャンネルのエンコーダーの Z 相信号の極性を一括で設定する。チャンネル毎には設定できない。ドライバに０渡すと、Z 相がアクティブ・ローに、１を渡すと、Z 相がアクティブ・ハイになる。

例　使うエンコーダーZ 相信号がアクティブ・ハイの場合、次のように書けばいい。

SendCmdDataEx(0, 0, 0x46, 0x0001);

(14) ENC_Set

このコマンドは指定したチャンネルのカウンタにカウント値を設定する。カウンタ値は24ビットなので、SendCmdDWDataEx によりコマンドを送信する。

例　チャンネル 3 のカウンタにカウント値 123456H を設定したい場合、次のように書けばいい。

SendCmdDWDataEx(0, 2, 0x47, 0x123456);

(15) PIO_Read

このコマンドは指定したポート・グループから 16Bit のデータを読み出す。３２チャンネルのポートは次のように二つのポート・グループに分けられる（表 3-3 をご参照)。このコマンドはチャンネルでなくグループ毎に指定するので、16 チャンネルのポートから同時にデータを読み出す。

表 3-3

ポート・グループ No	1	2
チャンネル範囲	1～16	17～32

例　チャンネル 17～32 のポートの値を読み出したい場合、次のように書けばいい。引数 iChNo にはグループ No-1 をセットする。

SendCmdEx(0, 1, 0x50);　ReadDataEx(0, &wData,1);

(16) PIO_Write

このコマンドは指定したポート・グループ（表 3-3 をご参照）に 16Bit のデータを書き出す。このコマンドはチャンネルでなくグループ毎に指定するので、16 チャンネルのポートに同時にデータを書き出すのでご注意下さい。1 チャンネル毎に出力したい場合にコマンド PIO_Set と PIO_Clear をご利用下さい。

初期状態では全ての出力ポートがローになっている。

例　チャンネル 17～32 のポートを全てハイにセットしたい場合、次のように書けばいい。引数 iChNo にはグループ No-1 をセットする。

SendCmdDataEx(0, 1, 0x51, 0xFFFF);

(17) PIO_Set

このコマンドは指定したチャンネルのポートをハイにセットする。

例　チャンネル 16 のポートをハイにセットしたい場合、次のように書けばいい。

SendCmdEx(0, 15, 0x52);

(18) PIO_Clear

このコマンドは指定したチャンネルのポートをローにセットする。

例　チャンネル 16 のポートをローにセットしたい場合、次のように書けばいい。

```
SendCmdEx(0, 15, 0x53);
```

(19) INT_Enable

このコマンドは指定したチャンネルの外部割込みを有効にする。その割込み入力端子にローからハイになる信号を与えると割込みが発生する。詳しくは「4．割込み仕様」をご参照下さい。

例　チャンネル 1 の外部割込みを許可したい場合、次のように書けばいい。

```
SendCmdEx(0, 0, 0x60);
```

(20) INT_Disable

このコマンドは指定したチャンネルの外部割込みを禁止する。その割込み入力端子に信号を与えても割込みが発生しない。

初期状態では全ての外部割込みが禁止状態になっている。

例　チャンネル 1 の外部割込みを禁止したい場合、次のように書けばいい。

```
SendCmdEx(0, 0, 0x61);
```

(21) INT_Mode

このコマンドは指定したチャンネルの外部割込みのトリガーモードを設定する。マイコンに渡すデータとトリガーモードとの関係は次のように示す。デフォルトモードはロー・レベル・センス・モードになっている。

表 3-4

渡すデータ	トリガーモード
0	ロー・レベル・センス・モード
1	立下りエッジ・トリガー・モード

例　チャンネル 2 の外部割込みを立下りエッジトリガーに設定したい場合、次のように書けばいい。

```
SendCmdEx(0, 1, 0x62);
```

(22) INT_Clear

このコマンドは RIB-PCI にある割込み発生フラグ INT をクリアする。フラグ INT をクリアしないと新しい割込みが入れなくなるので、PC 側が割込みを受け付けて処理を済ませたら必ずこのコマンドを送って INT をクリアしてください。

例　割込み発生フラグ INT をクリアしたい場合、次のように書けばいい。

```
SendCmdEx(0, 0, 0x64);
```

(22) ALL_Reset

このコマンドは RIB-PCI ボードをリセットし、電源を入れた直後の初期状態に戻させる。緊急の場合、又は異常が発生した場合、このコマンドを送信すれば全ての出力を停止できる。

例　RIB-PCI の全てのチャンネルの出力を停止したい場合、次のように書けばいい。

```
SendCmdEx(0, 0, 0xFF);
```

４．Linux ドライバ・ソフト仕様

最近、リアルタイム制御に最も相応しい OS として RT-Linux が注目されている。RIB-PCI は、お客様の強い要望に応えて、RT-Linux のドライバも提供する。但し、Linux ドライバは Windows ドライバ違って、ソースコードの形で提供している。従って、次の二点をご注意されたい。

●ドライバソフトを利用する前に、ソースコードをコンパイルして実行モジュールを作成し、システムにインストールする作業を行う必要がある。

●提供するソースコードをカスタマイズして利用することも可能であるが、カスタマイズにより生じる不具合などについてはサポートしかねるので、ご了承下さい。

一、Linux ドライバのソースファイル構成

ドライバのソースファイル一覧が次のようになっている。

ファイル名	役割説明
README	ドライバに関する説明（使用する前に必ず読んでください）
Makefile	RT-LINUX 用ドライバのコンパイルとインストール
JwRib.mk	標準 LINUX 用ドライバの作成時にインクルードするファイル
RibPciMod.c	ドライバ・モジュールのソースファイル
RibPciMod.h	ドライバ・モジュール用ヘッドファイル
Rib_Intr.c	ユーザーが作成する割込み処理ルーチンのソールファイル
RibDef.h	ドライバやその他のアプリケーションが利用する定義ファイル
Hardware.h	ハードウェアに関するヘッドファイル
RibApic.h	PCI コントロールに関するヘッドファイル
RibFunc.h	関数の宣言用ヘッドファイル
RibMain.c	RIB-PCI 仮想 I/O インタフェースのソースファイル
RibFunc.c	各種共通関数のソースファイル
RibApic.c	APIC コントロール制御関数のソースファイル
RibInt.c	割込み制御関数のソースファイル
RibAd.c	A/D コンバータ制御関数のソースファイル
RibDa.c	D/A コンバータ制御関数のソースファイル
RibEnc.c	エンコーダカウンター制御関数のソースファイル
RibPio.c	PIO 制御関数のソースファイル
RibPwm.c	PWM 制御関数のソースファイル

上記のファイルが全て付属のフロッピーディスクのフォルダ「/Linux/Driver」に格納されている。

二、Linux ドライバの構築方法

以下の手順に従ってドライバのソースコードをコンパイルしてインストールして下さい。

●まず、スーパーユーザーとしてログインする。
●付属 CD-ROM の Linux ディレクトリをそのままハードディスクのあるディレクトリにコピーして、Linux/Driver のディレクトリにカレントディレクトリにする。
●コマンド make clean を実行して、古いドライバを削除する。
●コマンド make を実行して、ソースファイルをコンパイルする。
●コマンド make dev を実行して、デバイスファイルを作成する。コマンド「ls/dev/jwribpci*」を実行してデバイスファイルが作られたことを確認して下さい。
●コマンド make insmod を実行して、ドライバをインストールする。
注意事項：
●ドライバごとに割り付けられる番号——メジャー番号は、デフォルトで９９になっている。この番号が既に使われている場合に、別の番号に変更する必要がある。そのとき、ファイル Makefile と RibPciMod.h の中身も一緒に変更しなければならない。
●インストールしたドライバを削除したい場合、コマンド make rmmod を実行して下さい。

三、Linux ドライバの利用方法

Windows ドライバと同じように、アプリケーションが簡単にインタフェースボードをアクセスするために、１０種類の API 関数を用意している。これらの API 関数の定義と宣言はそれぞれ付属の CD-ROM のフォルダ「/Linux/Sample」に格納されているファイル「Riblo.c」と「Riblo.h」に入っている。これらの API 関数を利用するアプリケーションは必ずこの二つのファイルと一緒にコンパイルしリンクしなければならない。以降、これらの API 関数について詳しく説明する。

1．ドライバ開始関数

関数名	**int OpenRibEx(int iRibNo)**
機　能	指定した番号の RIB-PCI ボードをアクセスするドライバを開始する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる）
戻り値	**>=0:** 正常終了　　**<0:** 異常終了

2．ドライバ終了関数

関数名	**int CloseRibEx(int iRibNo)**
機　能	指定した番号の RIB-PCI ボードをアクセスするドライバを終了する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる）
戻り値	**>=0:** 正常終了　　**<0:** 異常終了

3．アドレス取得関数

関数名	**int GetRegAdrEx(int iRibNo)**
機　能	RIB-PCI ボードの IO アドレスを取得する。

引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる）
戻り値	**>=0:** IO アドレス　　**<0:** 異常終了

4．割込み番号取得関数

関数名	**int GetIrqNoEx(int iRibNo)**
機　能	RIB-PCI が使う割込みの番号を取得する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる）
戻り値	**>=0:** 割込み番号　　**<0:** 異常終了

5．パラメータのないコマンド送信関数

関数名	**BOOL SendCmdEx(int iRibNo, int iChNo, WORD wCmd)**
機　能	パラメータを持っていないコマンドを RIB-PCI に送信する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **iChNo:** チャンネル番号（0 から始まる） **wCmd:** 送信するコマンド（第３章をご参考下さい）
戻り値	**TRUE:** 正常終了　　**FALSE:** 異常終了

6．パラメータのあるコマンド送信関数

関数名	**BOOL SendCmdDataEx(int iRibNo, int iChNo, WORD wCmd, WORD wData)**
機　能	パラメータを持っているコマンドを RIB-PCI に送信する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **iChNo:** チャンネル番号（0 から始まる） **wCmd:** 送信するコマンド（第３章をご参考下さい） **wData:** コマンドが持っているパラメータ
戻り値	**TRUE:** 正常終了　　**FALSE:** 異常終了

7．ダブルワード・パラメータのあるコマンド送信関数

関数名	**BOOL SendCmdDWDataEx(int iRibNo, int iChNo, WORD wCmd, DWORD dwData)**
機　能	ダブルワード・パラメータを持っているコマンドを RIB-PCI に送信する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **iChNo:** チャンネル番号（0 から始まる） **wCmd:** 送信するコマンド（第３章をご参考下さい） **dwData:** コマンドが持つダブルワード・パラメータ（例えばカウンターの値）
戻り値	**TRUE:** 正常終了　　**FALSE:** 異常終了

８．データ受信関数

関数名	**BOOL ReadDataEx(int iRibNo, WORD *pwData, int iCount)**
機　能	RIB-PCI からデータを受信する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **pwData:** 受信データを受取るバッファーのポイント。バッファーの長さは **iCount** ワード以上でなければならない。 **iCount:** 受信データのワード数
戻り値	**TRUE:** 正常終了　　**FALSE:** 異常終了

９．ダブルワード・データ受信関数

関数名	**BOOL ReadDWDataEx(int iRibNo, DWORD *pdwData)**
機　能	RIB-PCI からダブルワード・データを受信する。
引　数	**iRibNo:** RIB-PCI ボードの番号（0 から始まる） **pdwData:** ダブルワード・受信データ（例えばカウンターの値）を受取るバッファーのポイント。バッファーの長さはダブルワードである。
戻り値	**TRUE:** 正常終了　　**FALSE:** 異常終了

１０．ボード番号取得関数

関数名	**int GetRibNoEx(int iRibID)**
機　能	RIB-PCI ボードの番号を取得する。
引　数	**iRibID:** RIB-PCI ボードの ID（製造シリアル番号）
戻り値	**>=0:** ボード番号　　**<0:** 異常終了

補足説明：

●Windows ドライバと違って、Linux ドライバを利用するときに、必ず関数 OpenRibEx を呼び出してドライバを開始しなければならないし、利用後に必ず関数 CloseRibEx を呼び出してドライバを終了しなければならない。また、Linux ドライバでは、同時に使える RIB-PCI ボードの数を４枚に限るのでご注意頂きたい。

●Linux ドライバがソースファイルの形で提供されるので、割込み処理関数は CallBack 関数という形でなく、直接ドライバの中にある割込み処理関数の内容をカスタマイズして利用する形になる。割込み処理関数がソースファイル rib_intr.c に入っていて次のような形になっている。

```
static int rib_intr_routinue
   (int extintr,                  //  外部割込み番号(0-3)
    JWRIBISA_DEV *dev,   //  カレント dev struct へのポインタ
    struct pt_regs* regs)    //  プロセッサ・スナップショット(あまり使わない)
{
```

```
    // ここにユーザーが割込み処理プログラムを書き込む。
    // この割込み処理関数では時間のかかる処理を避けなければならない。
    return 0;
}
```

付属 CD-ROM のフォルダ「/Linux/Sample」にドライバが提供している API 関数の使い方を示すサンプルプログラムが格納されている。